Soft-max boosting

Matthieu Geist

Noname manuscript No.
(will be inserted by the editor)

Soft-max boosting

Matthieu Geist

Received: date / Accepted: date

Abstract The standard multi-class classification risk, based on the binary loss, is rarely directly minimized. This is due to *(i)* the lack of convexity and *(ii)* the lack of smoothness (and even continuity). The classic approach consists in minimizing instead a convex surrogate. In this paper, we propose to replace the usually considered deterministic decision rule by a stochastic one, which allows obtaining a smooth risk (generalizing the expected binary loss, and more generally the cost-sensitive loss). Practically, this (empirical) risk is minimized by performing a gradient descent in the function space linearly spanned by a base learner (a.k.a. boosting). We provide a convergence analysis of the resulting algorithm and experiment it on a bunch of synthetic and real-world data sets (with noiseless and noisy domains, compared to convex and non-convex boosters).

1 Introduction

The ideal multi-class classification task can be defined as the minimization (over some hypothesis space) of the expected binary loss (or equivalently, as minimizing the probability of predicting a wrong label). However, this risk is rarely directly minimized, notably because of *(i)* its lack of smoothness (and even continuity) and *(ii)* its lack of convexity. The classical approach consists in minimizing instead a convex (and possibly smooth) surrogate, such as the hinge loss (used in Support Vector Machines [23]), the exponential loss (used notably in AdaBoost [6], as shown for example in [17]) or the logit loss (for example used in [7]), among others. If using such surrogates does not come without guarantees [1], we propose an alternative approach in this article, focusing directly on the primary loss of interest.

Matthieu Geist
CentraleSupélec, IMS - MaLIS Research Group & UMI 2958 (GeorgiaTech-CNRS)
E-mail: matthieu.geist@centralesupelec.fr

The aforementioned classification methods are based on a deterministic decision rule. Here, we propose to use instead a stochastic decision rule. This does not provide a convex optimization problem, but at least we gain smoothness. The resulting (empirical) risk has then to be minimized. In order to ease the choice of the hypothesis space, we minimize this risk by performing a gradient descent in the function space linearly spanned by a base (or weak) learner. This is indeed a boosting [21] approach (seen as a functional gradient descent [17,9], which is not the sole nor the original view of boosting). To sum up, this article introduces a boosting approach that directly minimizes the expected (cost-sensitive) binary loss.

The paper is organized as follows. Sec. 2 states the problem and introduces and motivates the considered risk, that generalizes strictly the usual expected (cost-sensitive) binary loss. Sec. 3 shows how this risk can be minimized using a boosting approach and introduces the related algorithm, named sm-boost (for soft-max boosting). The proposed method is analyzed in Sec. 4. We show mainly that the algorithm converges to a solution such that the related functional gradient is null, at a given rate. We also discuss informally the quality of the related solution (easy formal response being not available due to the lack of convexity). In Sec. 5, we discuss the choice of meta-parameters for sm-boost and present briefly AdaBoost.MH and SAMME, two efficient multi-class generalizations of AdaBoost which are natural competitors to sm-boost. We then compare all algorithms on a bunch of synthetic and real-world data sets. We also compare sm-boost to another non-convex booster, MartiBoost, on a challenging (for convex boosters) noisy toy problem. Eventually, we open some perspectives in Sec. 6.

2 Problem statement

Let $\mathcal{D}_N = \{(x_i, y_i)_{1 \leq i \leq N}\}$ be a data set, with $x_i \in \mathcal{X}$, a compact subset of $\mathbb{R}^d$, and $y_i \in \mathcal{Y}$, a finite set of labels. The inputs x_i are sampled according to some unknown distribution $\rho(x)$ and the outputs y_i are provided by an oracle (formally an unknown conditional distribution $\mathbb{P}(y|x)$). Let $c \in \mathbb{R}_+^{\mathcal{X} \times \mathcal{Y}^2}$ be a (bounded) cost function: $c(x_i, y_i, y)$ is the cost (or loss) for choosing the label y instead of the oracle response y_i for the input x_i. This encompasses notably the classical binary loss, on which we will focus in the experimental section (Sec. 5):

$$c(x_i, y_i, y) = \mathbb{I}_{\{y \neq y_i\}} = \begin{cases} 1 \text{ if } y \neq y_i \\ 0 \text{ else} \end{cases} .$$

Let $\mathcal{G}$ be a subset of (deterministic) functions mapping inputs to labels (that is, deterministic decision rules), $\mathcal{G} \subset \mathcal{Y}^{\mathcal{X}}$. The "ideal" (possibly cost-sensitive) multi-class classification problem may be defined as the minimization of the

risk related to the above defined cost:

$$\min_{g\in\mathcal{G}} R(g) \text{ with } R(g) = \int_{\mathcal{X}} \sum_{y\in\mathcal{Y}} c(x,y,g(x))\mathbb{P}(y|x)\rho(x)dx$$
$$= \mathbb{E}_{X,Y}[c(X,Y,g(X))]. \quad (1)$$

The input and oracle distributions being only known through the data set, practically the related empirical risk is considered:

$$R_N(g) = \frac{1}{N}\sum_{i=1}^{N} c(x_i,y_i,g(x_i)).$$

Notably, specialized to the binary loss, this gives:

$$R_N(g) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_{\{y_i\neq g(x_i)\}}.$$

There are two major issues with this risk. First, it is not convex, many local minimum may exist. Second, it is not smooth (or even continuous here), which may prevent to practically minimize it. The classical approach to overcome this problem consists in minimizing a convex (and possibly smooth) surrogate to this risk (for example, exponential or logit loss instead of the binary one). Here, we propose an alternative approach: we replace the deterministic decision rule by a stochastic one, parameterized through a soft-max distribution. By doing so, we do not gain convexity, but we gain smoothness. We also strictly generalize the "ideal" risk (deterministic decision rules being a special case of stochastic ones). What we argue in the sequel is that minimizing directly the risk of interest is a viable alternative to standard approaches.

2.1 Proposed risk

Let Ψ be a subset of $\mathbb{R}^{\mathcal{X}\times\mathcal{Y}}$. A function $\psi\in\Psi$ can be understood as a score function, ranking different labels for a given input. In classification, it is quite usual to define a deterministic decision rule as $g_\psi(x) = \operatorname{argmax}_{y\in\mathcal{Y}} \psi(x,y)$. Here, we propose instead to build a stochastic decision rule based on a soft-max conditional distribution. Let $\Delta_{\mathcal{Y}}$ be the set of distributions over $\mathcal{Y}$. For $\psi\in\Psi$, we define the decision rule $g_\psi \in \Delta_{\mathcal{Y}}^{\mathcal{X}}$ as:

$$g_\psi(y|x) = \frac{e^{\psi(x,y)}}{\sum_{z\in\mathcal{Y}} e^{\psi(x,z)}}. \quad (2)$$

Then, what we propose to do is to consider directly the ideal risk provided in Eq. (1), and to minimize it over stochastic decision rules of the form given in

Eq. (2), instead of minimizing it over deterministic decision rules:

$$\begin{aligned} R(g_\psi) &= \mathbb{E}_{X,Y}[\mathbb{E}_{Z\sim g_\psi(.|X)}[g_\psi(Z|X)c(X,Y,Z)]] \\ &= \mathbb{E}_{X,Y}[\sum_{z\in\mathcal{Y}} g_\psi(z|X)c(X,Y,z)] \\ &= \int_{\mathcal{X}} \sum_{y\in\mathcal{Y}}\sum_{z\in\mathcal{Y}} c(x,y,z)g_\psi(z|x)\mathbb{P}(y|x)\rho(x)dx. \end{aligned} \tag{3}$$

As the input and oracle distributions are only known through the data, we consider the following empirical risk:

$$\begin{aligned} \min_{\psi\in\Psi} R_N(\psi) \text{ with } R_N(\psi) &= \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{y\sim g_\psi(.|x_i)}[c(x_i,y_i,y)] \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{y\in\mathcal{Y}} g_\psi(y|x_i)c(x_i,y_i,y). \end{aligned} \tag{4}$$

2.2 Motivation

As said before, the common technique in classification consists in minimizing a convex surrogate to the risk of interest. The resulting optimization problem becomes tractable, but one may wonder if minimizing such a convex proxy still results in a good accuracy. This is known as the problem of *calibration*, which studies how small the suboptimality gap, measured in term of the surrogate risk, should be to achieve a suboptimality gap, measured in term of the risk of interest, of a given size. This has been studied in the binary cost-insensitive case [1], in the binary cost-sensitive case [24] and in the cost-sensitive multi-classe case [20], for the convex surrogate proposed in [14].

However, it has also been advocated recently that a surrogate loss function should be *guess-averse* [2], in the sense that the loss should encourage more correct classifications than arbitrary guesses. If all popular binary surrogate losses are guess averse, things become less clear in the cost-sensitive multi-class case. Notably, it shown in [2] that the convex surrogate of [14] based on the exponential function, shown to be calibrated in [20], is not guess averse (and the question remains open for this surrogate based on other convex functions).

To sum up, choosing the right convex surrogate to the cost-sensitive multi-class risk of Eq. (1) is no an easy task. Moreover, in the case of boosting, it has been shown that convex boosters are necessarily sensitive to noise [16] (this point will be further addressed in Sec. 5.5). What we propose here is to minimize directly the risk of interest, given in Eq. (1), but parameterized by the stochastic decision rule of Eq. (2) instead of the classical deterministic one, which provides the risk of Eq. (3). As deterministic decision rules are a special case of stochastic one, this is a generalization of the risk of interest rather than a surrogate. Moreover, the resulting optimization problem is smooth (see Sec. 4), and is thus tractable.

Yet, one may still wonder if this changes the problem at hand. Notably, as the decision rule is stochastic, one may think that risk (3) ultimately estimates conditional probabilities (of labels conditioned on inputs) and thus that it is a kind of logistic regression. This is not the case: ultimately, risk (3) provides the same solution as risk (1), as formalized below.

Proposition 1 *Assume that the cost function corresponds to a single label problem, in the sense that:*

$$\mathrm{Card}\left(\operatorname*{argmin}_{y\in\mathcal{Y}} \mathbb{E}[c(X,Y,y)|X]\right) = 1 \text{ almost surely.}$$

Assume also that there exists a measurable function $g^* \in \mathcal{Y}^{\mathcal{X}}$ such that

$$g^*(x) = \operatorname*{argmin}_{y\in\mathcal{Y}} \mathbb{E}[c(X,Y,y)|X=x],$$

which is the minimizer of $R(g) = \mathbb{E}_{X,Y}[c(X,Y,g(X))]$ over $g \in \mathcal{Y}^{\mathcal{X}}$. Then, the minimizer of $R(g_\psi) = \mathbb{E}_{X,Y}[\sum_{z\in\mathcal{Y}} g_\psi(z|X)c(X,Y,z)]$ over $g_\psi \in \Delta_{\mathcal{Y}}^{\mathcal{X}}$ is:

$$g_\psi^*(y|x) = \mathbb{I}_{\{y=g^*(x)\}}.$$

Proof Assume that R admits a minimizer $g_\psi \neq g_\psi^*$. Write $c(x,y)$ the expected cost for choosing y in x, $c(x,y) = \mathbb{E}[c(X,Y,y)|X=x]$, and U the set of points where the minimizers disagree:

$$U = \{x \in \mathcal{X} | g_\psi(.|x) \neq g_\psi^*(.|x)\}.$$

If $\mathbb{P}(U) = 0$, the minimizers are equal almost surely. Else, for all $x \in U$ we have:

$$\begin{aligned}
\sum_{z\in\mathcal{Y}} c(x,z)g_\psi(z|x) &= c(x,g^*(x))g_\psi(g^*(x)|x) + \sum_{z\neq g^*(x)} c(x,z)g_\psi(z|x) \\
&> c(x,g^*(x))\{g_\psi(g^*(x)|x) + \sum_{z\neq g^*(x)} g_\psi(z|x)\} \\
&= c(x,g^*(x)) = \sum_{z\in\mathcal{Y}} c(x,z)g^*\psi(z|x) \text{ almost surely.}
\end{aligned}$$

Therefore, $R(g_\psi) > R(g_\psi^*)$ and g_ψ is not a minimizer. □

We have just shown that, in the single label case, minimizing the risk over stochastic decision rules does not change the solution. In the multi-label case, the stochastic minimizer can distribute its probability masses over minimizers of the expected cost $\mathbb{E}[c(X,Y,z)|X=x]$ without changing the risk, which is better than choosing a single label (as would do the deterministic minimizer).

Now that we have motivated the use of this risk, it will be shown how its empirical counterpart can be minimized and what guarantees it might offer.

3 A boosting approach

In this section, we minimize the risk defined in Eq. (4) using a boosting approach. To do so, we frame boosting as a functional gradient descent [17] to derive the proposed sm-boost (soft-max boosting) algorithm. First, we introduce the relevant related Hilbert space, following [9].

3.1 Relevant Hilbert space

With a slight abuse of notations, we will see equivalently a function from $\mathbb{R}^{\mathcal{X}\times\mathcal{Y}}$ as a function from $(\mathbb{R}^{\mathcal{Y}})^{\mathcal{X}}$. Notably, we will write:

$$\psi(x) = (\psi(x,y))_{y\in\mathcal{Y}} \in \mathbb{R}^{\mathcal{Y}}$$
$$\text{and } g_\psi(x) = (g_\psi(y|x))_{y\in\mathcal{Y}} \in \Delta_{\mathcal{Y}} \subset \mathbb{R}^{\mathcal{Y}}.$$

Let $\langle .,. \rangle$ be the Euclidian inner product in $\mathbb{R}^{\mathcal{Y}}$ and $\|.\|$ be the associated norm. Assume that the input space $\mathcal{X}$ is measurable and let μ be a probability measure. The function space $L^2(\mathcal{X}, \mathbb{R}^{\mathcal{Y}}, \mu)$ is the set of all equivalence classes of functions $\psi \in (\mathbb{R}^{\mathcal{Y}})^{\mathcal{X}}$ such that the Lebesgue integral $\int_{\mathcal{X}} \|\psi(x)\|^2 d\mu$ is finite. This Hilbert space has a natural inner product:

$$\begin{aligned}\langle \psi, \phi \rangle_\mu &= \int_{\mathcal{X}} \langle \psi(x), \phi(x) \rangle d\mu \\ &= \int_{\mathcal{X}} \sum_{y\in\mathcal{Y}} \psi(x,y)\phi(x,y) d\mu.\end{aligned}$$

As the quantity of interest is the empirical risk, the natural probability measure to be considered is $\hat{\rho}$, involved by the data set $\mathcal{D}_N$. We write $\langle .,. \rangle_N$ the related inner product, defined as:

$$\begin{aligned}\langle \psi, \phi \rangle_N &= \frac{1}{N}\sum_{i=1}^N \langle \psi(x_i), \phi(x_i) \rangle \\ &= \frac{1}{N}\sum_{i=1}^N \sum_{y\in\mathcal{Y}} \psi(x_i,y)\phi(x_i,y).\end{aligned}$$

We also adopt the following notation for the cost c, evaluated in the training points:

$$c_i = (c(x_i, y_i, y))_{y\in\mathcal{Y}} = (c_i(y))_{y\in\mathcal{Y}} \in \mathbb{R}^{\mathcal{Y}}. \tag{5}$$

This way, the considered empirical risk (defined in Eq. (4)) can be written as follows:

$$R_N(\psi) = \frac{1}{N}\sum_{i=1}^N \langle c_i, g_\psi(x_i) \rangle = \frac{1}{N}\sum_{i=1}^N \sum_{y\in\mathcal{Y}} c_i(y) g_\psi(y|x_i).$$

We will compute a decision rule g_ψ by performing a gradient descent on $R_N(\psi)$ (according to ψ).

However, to do so, one has to compute the gradient of functionals over the Hilbert space $L^2(\mathcal{X}, \mathbb{R}^\mathcal{Y}, \hat{\rho})$. We consider here the Fréchet derivative. Let $J : L^2(\mathcal{X}, \mathbb{R}^\mathcal{Y}, \hat{\rho}) \to \mathbb{R}$ be a functional (practically, it will be R_N), its (Fréchet) derivative is the linear operator $\nabla J(\psi)$ satisfying

$$\lim_{\phi \to 0} \frac{\|J(\psi + \phi) - J(\psi) - \langle \nabla J(\psi), \phi \rangle_N\|}{\|\phi\|_N} = 0.$$

It can also be implicitly defined as

$$J(\psi + \phi) = J(\psi) + \langle \nabla J(\psi), \phi \rangle_N + o(\|\phi\|_N).$$

3.2 Restricted gradient descent

The issue with functions representing the functional gradient is that they are hard to manipulate and do not generalize well to new inputs. Instead of being directly followed, the gradient is projected on a restriction set of allowable directions (practically, the set of hypotheses generated by a weak learner). This is termed as restricted gradient descent in [9].

Let $\mathcal{H} \subset \mathbb{R}^{\mathcal{X} \times \mathcal{Y}}$ be the hypothesis space generated by a weak learner. The closest function h' to a gradient $\nabla J(\psi)$ along a candidate direction h is given by the vector projection:

$$h' = \frac{\langle \nabla J(\psi), h \rangle_N}{\|h\|_N^2} h.$$

The function h^* minimizing the resulting projection error actually maximises the projected length:

$$h^* = \operatorname*{argmax}_{h \in \mathcal{H}} \frac{\langle \nabla J(\psi), h \rangle_N}{\|h\|_N}. \tag{6}$$

This projection operation, proposed in [9], generalizes the one of [17] to functions other than classifiers (yet, this is the case we will consider practically).

Boosting can be seen as a gradient descent in a function space, the functional gradient being projected on the hypothesis space generated by a base (or weak) learner. This is summarized in Alg. 1 for the minimization of a functional J.

3.3 The softmax boosting algorithm

The functional we are interested in here is $R_N(\psi)$, provided in Eq. (4). A first thing to do is to compute its functional gradient, which is done below.

Algorithm 1: Basic boosting algorithm

Inputs;
Initial function ψ_0 ;
Learning rates $(\alpha_t)_{t>0}$;
Hypothesis space $\mathcal{H}$;
for $t = 1, 2, \dots T$ **do**
  Compute the gradient $\nabla_t = \nabla J(\psi_{t-1})$;
  Project ∇_t onto $\mathcal{H}$ (see Eq. (6)), finding nearest directions h_t^* ;
  Update $\psi : \psi_t = \psi_{t-1} - \alpha_t \frac{\langle h_t^*, \nabla_t \rangle_N}{\|h_t^*\|_N^2} h_t^*$;

Proposition 2 *Let $e_y \in \mathbb{R}^{\mathcal{Y}}$ be a vector such that all components are set to 0, except the one corresponding to y which is set to 1. Define $\Delta_\psi c_i(y)$ as the centered cost according to the probability measure g_ψ induced by ψ, for the input x_i (the notation $c_i(y)$ being defined in Eq. (5)):*

$$\Delta_\psi c_i(y) = c_i(y) - \sum_{z \in \mathcal{Y}} g_\psi(z|x_i) c_i(z) = c_i(y) - \mathbb{E}_{z \sim g_\psi(.|x_i)}[c_i(z)].$$

The functional gradient of $R_N(\psi)$ is the set

$$\nabla R_N(\psi) = \left\{ \phi \in (\mathbb{R}^{\mathcal{Y}})^{\mathcal{X}} : \phi(x_i) = \sum_{y \in \mathcal{Y}} g_\psi(y|x_i) \Delta_\psi c_i(y) e_y \right\}.$$

Proof Recall that the considered gradient is defined according to the inner product under the empirical measure $\hat{\rho}$. Therefore, only the values taken in the data points x_i do matter. Let $\phi \in \nabla R_N(\psi)$, we have:

$$\begin{aligned} \phi(x_i) &= \nabla \langle c_i, g_\psi(x_i) \rangle \\ &= \sum_{y \in \mathcal{Y}} c_i(y) \nabla g_\psi(y|x_i). \end{aligned}$$

Using a log-trick and the definition of g_ψ (see Eq. (2)):

$$\begin{aligned} \nabla g_\psi(y|x) &= g_\psi(y|x) \nabla \ln g_\psi(y|x) \\ &= g_\psi(y|x) \nabla \{ \psi(x,y) - \ln \sum_{z \in \mathcal{Y}} e^{\psi(x,z)} \} \\ &= g_\psi(y|x) (\nabla \psi(x,y) - \frac{\sum_{z \in \mathcal{Y}} e^{\psi(x,z)} \nabla \psi(x,z)}{\sum_{z \in \mathcal{Y}} e^{\psi(x,z)}}) \\ &= g_\psi(y|x) (\nabla \psi(x,y) - \sum_{z \in \mathcal{Y}} g_\psi(z|x) \nabla \psi(x,z)). \end{aligned}$$

Therefore, we have:

$$\begin{aligned}
\phi(x_i) &= \sum_{y\in\mathcal{Y}} c_i(y) g_\psi(y|x_i)(\nabla\psi(x_i,y) - \sum_{z\in\mathcal{Y}} g_\psi(z|x_i)\nabla\psi(x_i,z)) \\
&= \sum_{y\in\mathcal{Y}} c_i(y) g_\psi(y|x_i)(e_y - \sum_{z\in\mathcal{Y}} g_\psi(z|x_i) e_z) \\
&= \sum_{y\in\mathcal{Y}} c_i(y) g_\psi(y|x_i) e_y - (\sum_{y\in\mathcal{Y}} c_i(y) g_\psi(y|x_i))(\sum_{z\in\mathcal{Y}} g_\psi(z|x_i) e_z) \\
&= \sum_{y\in\mathcal{Y}} g_\psi(y|x_i) e_y (c_i(y) - \sum_{y\in\mathcal{Y}} c_i(y) g_\psi(y|x_i)) \\
&= \sum_{y\in\mathcal{Y}} g_\psi(y|x_i) e_y \Delta_\psi c_i(y).
\end{aligned}$$

This concludes the proof. □

Thanks to this result, the inner product between a function $h \in \mathcal{H}$ and the functional gradient $\nabla R_N(\psi)$ is given by:

$$\begin{aligned}
\langle \nabla R_N(\psi), h\rangle_N &= \frac{1}{N}\sum_{i=1}^{N} \langle \phi(x_i), h(x_i)\rangle, \text{ with } \phi \in \nabla R_N(\psi) \\
&= \frac{1}{N}\sum_{i=1}^{N}\sum_{y\in\mathcal{Y}} g_\psi(y|x_i)\Delta_\psi c_i(y) h(x_i,y).
\end{aligned}$$

For the rest of this paper, we will focus on hypothesis spaces generated by binary classifiers as weak learners, that is:

$$\mathcal{H} \subset \{-1,+1\}^{\mathcal{X}\times\mathcal{Y}}.$$

Let $K = |\mathcal{Y}|$ denote the cardinal of the finite set $\mathcal{Y}$. For binary classifiers, we have:

$$\forall h \in \mathcal{H}, \quad \|h\|_N^2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{y\in\mathcal{Y}} h(x_i,y)^2 = K.$$

Therefore, projecting the gradient onto $\mathcal{H}$ reduces to maximizing $\langle \nabla R_N(\psi), h\rangle$:

$$h^* \in \operatorname*{argmax}_{h\in\mathcal{H}} \frac{1}{N}\sum_{i=1}^{N}\sum_{y\in\mathcal{Y}} g_\psi(y|x_i)\Delta_\psi c_i(y) h(x_i,y).$$

Consequently, using the proposed approach (projected functional gradient descent on R_N), the initial (possibly cost-sensitive) multi-class classification problem is reduced to a series of weighted binary classification problems.

Yet, it has been shown that introducing randomization in a boosting procedure can improve the estimation accuracy [8]. The inner product to be maximized is indeed an expectation:

$$\begin{aligned}\langle \nabla R_N(\psi), h\rangle_N &= \frac{1}{N}\sum_{i=1}^{N}\sum_{y\in\mathcal{Y}} g_\psi(y|x_i)\Delta_\psi c_i(y)h(x_i,y) \\ &= \mathbb{E}_{x,y\sim\hat{\rho}(x)g_\psi(y|x)}[\Delta_\psi c_i(y)h(x_i,y)]. \end{aligned} \tag{7}$$

So, to introduce randomization, we propose to sample M inputs $\{(x_j,z_j)_{1\leq j\leq M}\}$ from the distribution $\hat{\rho}(x)g_\psi(z|x)$ (the probability of a couple (x_j,z_j) being thus $\frac{1}{N}g_\psi(z_j|x_j)$). The candidate direction is then computed as:

$$h^* \in \operatorname*{argmax}_{h\in\mathcal{H}} R_M(h) \text{ with } R_M(h) = \frac{1}{M}\sum_{j=1}^{M}\Delta_\psi c_j(z_j)h(x_j,z_j). \tag{8}$$

This is a weighted binary classification problem with inputs $(x_j,y_j)_{1\leq j\leq M}$, weights $|\Delta_\psi c_j(z_j)|$ and outputs $\operatorname{sgn}(\Delta_\psi c_j(z_j))$.

Algorithm 2: Softmax boosting (sm-boost)

Inputs;
Data set $\{(x_i,y_i)_{1\leq i\leq N}\}$;
Weak learner ($\mathcal{H}$) ;
Learning rates $(\alpha_k)_{k>0}$;
Number of samples to be drawn for projection M ;

Initialization;
Initialize $\psi_0 = 0$;

for $t = 1, 2, \ldots T$ **do**

Draw M **inputs** for the weighted binary classif. ;
$(x_j,z_j)_{1\leq j\leq M} \sim \hat{\rho}(x)g_{\psi_{t-1}}(z|x)$;

Solve h_t^* as the solution of the weighted binary classification problem with ;
inputs: $(x_j,z_j)_{1\leq j\leq M}$;
weights: $(|\Delta_{\psi_{t-1}}c_j(z_j)|)_{1\leq j\leq M}$;
outputs: $(\operatorname{sgn}(\Delta_{\psi_{t-1}}c_j(z_j))_{1\leq j\leq M}$;

Update the score function ;
$\psi_t = \psi_{t-1} - \alpha_t \frac{\langle \nabla R_N(\psi_{t-1}), h_t^*\rangle_N}{\|h_t^*\|_N^2} h_t^*$;

Output the final hypothesis g_{ψ_T} ;

The proposed approach is summarized in Alg. 2. Choosing $\psi_0 = 0$ is a natural default initialization, as it corresponds to a uniform decision rule (equiprobability for each label). There are three meta-parameters: the weak learner ($\mathcal{H}$), the learning rates ($(\alpha_t)_{t>0}$) and the number of samples to be drawn at each iteration (M). We will discuss them in Sec. 5, partly based on the analysis

to be done in the next section (Sec. 4). Before, it may be useful to provide a quick and intuitive explanation of the rationale behind the algorithm.

At step t, the algorithm has already computed a stochastic decision rule $g_{\psi_{t-1}}(y|x)$. For a given input x_i, using this decision rule, we pay an average cost (or loss) of $\mathbb{E}_{y\sim g_{\psi_{t-1}}(.|x_i)}[c_i(y)]$. For a given label y, if the cost $c_i(y)$ is higher than the expected cost, then the binary classifier will have a target of $+1$, with a weight proportional to the absolute difference between the label cost and the payed averaged cost (that is, $|\Delta_{\psi_{t-1}} c_j(y)|$). Ideally, we have $h_t^*(x_i, y) = +1$. Given the update rule, this means that the probability of choosing this label will decrease in the next decision rule, g_{ψ_t}. Similarly, if the cost $c_i(y)$ is lower than the expected cost, then the probability of choosing this label will increase.

We also discuss briefly the computational cost of this algorithm. At each iteration, the decision rule must be updated, for all labels and inputs of the data set, which can be done in $O(NK)$. Drawing the M inputs can be done by performing a binary search of M random samples (uniform distribution) in the cumulative sum of computed probabilities, which has a cost of $O(NK + M\log(NK))$. Computing the weights and outputs used by the base learners has a cost of $O(MK)$. The cost for training the weak learner depends on the weak learner. For example, constructing a decision tree (the base learner considered in this paper) with M samples has roughly a cost of $O(M\log(M))$ [19]. Here, we train K weak learners, each with M_i data points (satisfying $\sum_{i=1}^K M_i = M$). As $\sum_{i=1}^K M_i \log(M_i) = M\log(M) + \sum_{i=1}^K M_i \log(\frac{M_i}{M}) \leq M\log(M)$, the cost would be of $O(M\log(M))$ (practically, less than training one classifier with all data points). This gives a rough total cost per iteration of $O(NK + M\log(MNK))$.

4 Analysis

Using a stochastic decision rule allows gaining some smoothness, as shown in the next lemma. However, it does not transform the related optimization problem (minimization of $R_N(\psi)$) into a convex one. Smoothness will allow showing that the norm of the functional gradient tends to 0 (at a given rate), but the lack of convexity prevents from providing easily stronger results. Nevertheless, we will also discuss informally the quality of the computed solution.

Lemma 1 *Without loss of generality, assume that* $c \in [0,1]^{\mathcal{X}\times\mathcal{Y}^2}$. *The functional* $R_N(\psi)$ *defined in Eq.* (4) *is* 1*-Lipschitz-smooth:*

$$\forall \psi, \phi \in L^2(\mathcal{X}, \mathbb{R}^{\mathcal{Y}}, \hat{\rho}), \quad \|\nabla R_N(\psi) - \nabla R_N(\phi)\|_N \leq \|\psi - \phi\|_N.$$

Proof To prove this result, we will first study the function $F \in (\mathbb{R})^{\mathbb{R}^K}$ defined as

$$F(u) = \sum_{k=1}^K \alpha_k f_k(u) \text{ with } \alpha_k \in [0,1] \text{ and } f_k(u) = \frac{e^{u^\top e_k}}{\sum_{l=1}^K e^{u^\top e_l}}.$$

We will show that F is 1-Lipschitz-smooth. To do so, we first compute its gradient (resp. to $u \in \mathbb{R}^K$). Similarly to the proof of Prop. 2, we have:

$$\begin{aligned}\nabla F(u) &= \sum_{k=1}^{K} f_k(u) \left(\alpha_k - \sum_{l=1}^{K} \alpha_l f_l(u) \right) e_k \\ &= \sum_{k=1}^{K} f_k(u) \left(\alpha_k - F(u) \right) e_k.\end{aligned}$$

Next, we compute its Hessian $H(u) = \nabla \nabla^\top F(u)$:

$$\begin{aligned}H(u) &= \nabla \{ \sum_{k=1}^{K} f_k(u)(\alpha_k - F(u)) e_k^\top \} \\ &= \sum_{k=1}^{K} (\alpha_k - F(u))(\nabla f_k(u)) e_k^\top - \sum_{k=1}^{K} f_k(u)(\nabla F(u)) e_k^\top \\ &= \sum_{k=1}^{K} (\alpha_k - F(u)) f_k(u)(e_k - \sum_{l=1}^{K} f_l(u) e_l) e_k^\top \\ &\qquad - \sum_{k=1}^{K} f_k(u)(\sum_{l=1}^{K} f_l(u)(\alpha_l - F(u)) e_l) e_k^\top \\ &= \sum_{k=1}^{K} (\alpha_k - F(u)) f_k(u)(1 - f_k(u)) e_k e_k^\top - \sum_{k=1}^{K} (\alpha_k - F(u)) f_k^2(u) e_k e_k^\top \\ &= \sum_{k=1}^{K} f_k(u)(1 - 2 f_k(u))(\alpha_k - F(u)) e_k e_k^\top.\end{aligned}$$

From this expression, one can see that the convexity of F is far from being guaranteed (H being a diagonal matrix with possibly negative elements). Let $v \in \mathbb{R}^K$, we have

$$\begin{aligned}\|H(u)v\|^2 &= v^\top H(u)^\top H(u) v \\ &= \sum_{k=1}^{K} f_k^2(u)(1 - 2 f_k(u))^2 (\alpha_k - F(u))^2 v_i^2 \leq \sum_{k=1}^{K} v_i^2 = \|v\|^2.\end{aligned}$$

Therefore, the induced norm of the Hessian satisfies $\|H(u)\| \leq 1$, for all $u \in \mathbb{R}^K$. This shows that F is 1-Lipschitz-smooth, so

$$\begin{aligned}\|\nabla F(u) - \nabla F(v)\| &= \| \sum_{k=1}^{K} \{ f_k(u)(\alpha_k - F(u)) - f_k(v)(\alpha_k - F(v)) \} e_k \| \\ &\leq \|u - v\|.\end{aligned}$$

Going back to the functional of interest, we have (through Prop. 2):

$$\|\nabla R_N(\psi)-\nabla R_N(\phi)\|_N^2 = \frac{1}{N}\sum_{i=1}^{N}\|\sum_{y\in\mathcal{Y}}\{g_\psi(y|x_i)\Delta_\psi c_i(y)-g_\phi(y|x_i)\Delta_\phi c_i(y)\}e_y\|^2.$$

Thanks to the preceding result (u plays the role of $\psi(x_i)$, v of $\phi(x_i)$, α_y of $c_i(y)$), we have for all $1 \leq i \leq N$:

$$\|\sum_{y\in\mathcal{Y}}\{g_\psi(y|x_i)\Delta_\psi c_i(y) - g_\phi(y|x_i)\Delta_\phi c_i(y)\}e_y\| \leq \|\psi(x_i) - \phi(x_i)\|.$$

Therefore:

$$\|\nabla R_N(\psi) - \nabla R_N(\phi)\|_N^2 \leq \frac{1}{N}\sum_{i=1}^{N}\|\psi(x_i) - \phi(x_i)\|^2 = \|\psi - \phi\|_N^2.$$

This concludes the proof. □

This smoothness result is a key to show that sm-boost converges to a solution such that the (functional) gradient is null. For that, we must do some assumption on the capability of the hypothesis space $\mathcal{H}$ to approximate well the projected gradient.

Definition 1 The hypothesis space $\mathcal{H}$ has an edge $\gamma \in]0,1[$ if for every gradient $\nabla R_N(\psi)$, there exists a function $h \in \mathcal{H}$ such that:

$$\langle \nabla R_N(\psi), h\rangle_N \geq \gamma \|\nabla R_N(\psi)\|_N \|h\|_N.$$

This definition of edge has been introduced in [9]. For example, if $\mathcal{H}$ is a space of binary classifier that is able to do (strictly) better than random guessing for the provided data set and for any associated weights, then $\mathcal{H}$ has an edge $\gamma > 0$ (see Th. 1 and related discussion in [9] for more details).

The proposed convergence result can now be provided.

Theorem 1 *Assume that $\mathcal{H}$ has an edge $\gamma > 0$. Without loss of generality, assume also that $c \in [0,1]^{\mathcal{X}\times\mathcal{Y}^2}$. Let $(\psi_t)_{t\geq 0}$ be the sequence of functions computed by the sm-boost algorithm for the learning rate $\alpha_t = 1$ (for all $t > 0$). We have that:*

$$\lim_{t\to\infty}\|\nabla R_N(\psi_t)\|_N = 0.$$

Moreover, we have the following convergence rate:

$$\min_{1\leq t\leq T+1}\|\nabla R_N(\psi_t)\|_N \leq \frac{1}{\gamma}\sqrt{\frac{2}{T}}.$$

Proof The proof is basically an adaptation of the convergence of standard gradient descent for a smooth minimized function (*e.g.*, see [18]). Thanks to the smoothness result of Lemma 1, we have that for any $\psi, \phi \in L^2(\mathcal{X}, \mathbb{R}^{\mathcal{Y}}, \hat{\rho})$ (again, see [18] for example):

$$|R_N(\phi) - R_N(\psi) - \langle \nabla R_N(\psi), \phi - \psi\rangle_N| \leq \frac{1}{2}\|\phi - \psi\|_N^2.$$

Notably, for ψ_t and ψ_{t-1}:

$$R_N(\psi_t) \leq R_N(\psi_{t-1}) + \langle \nabla R_N(\psi_{t-1}), \psi_t - \psi_{t-1} \rangle_N + \frac{1}{2} \|\psi_t - \psi_{t-1}\|_N^2.$$

From the update rule, we have:

$$\psi_t - \psi_{t-1} = -\alpha_t \frac{\langle \nabla R_N(\psi_{t-1}), h_t^* \rangle_N}{\|h_t^*\|_N^2} h_t^*.$$

Therefore, we have:

$$R_N(\psi_t) \leq R_N(\psi_{t-1}) - \alpha_t (1 - \frac{\alpha_t}{2}) \left(\frac{\langle \nabla R_N(\psi_{t-1}), h_t^* \rangle_N}{\|h_t^*\|_N} \right)^2.$$

The quantity $\alpha_t(1 - \frac{\alpha_t}{2})$ is maximized for $\alpha_t = 1$ and the edge assumption implies that $\langle \nabla R_N(\psi_{t-1}), h_t^* \rangle_N \geq \gamma \|\nabla R_N(\psi_{t-1})\|_N \|h_t^*\|_N$. Therefore, we have:

$$\begin{aligned} & R_N(\psi_t) \leq R_N(\psi_{t-1}) - \frac{\gamma^2}{2} \|\nabla R_N(\psi_{t-1})\|_N^2 \\ \Leftrightarrow\ & R_N(\psi_{t-1}) - R_N(\psi_t) \geq \frac{\gamma^2}{2} \|\nabla R_N(\psi_{t-1})\|_N^2. \end{aligned}$$

Summing this last inequality over t, we get

$$\frac{\gamma^2}{2} \sum_{t=0}^{T-1} \|\nabla R_N(\psi_t)\|_N^2 \leq R_N(\psi_0) - R_N(\psi_T) \leq 1.$$

The last (rightmost) inequality results from the assumption that the cost is positive and bounded by 1. This allows concluding with bot results:

$$\lim_{t \to \infty} \|\nabla R_N(\psi_t)\|_N = 0 \text{ and } \min_{1 \leq t \leq T+1} \|\nabla R_N(\psi_t)\|_N \leq \frac{1}{\gamma} \sqrt{\frac{2}{T}}.$$

This concludes the proof. □

This result tells us that sm-boost will converge to a decision rule such that the functional gradient is null. However, it is rather weak as it does not tell anything about the quality of the solution. We can end up in a local minima, or even just in a stationary point. Indeed, without convexity, it might be difficult to tell more. Moreover, this result does not tell anything about the generalization capability of the computed estimator (a question we will address experimentally in this paper, but not theoretically).

Yet, some informal insights regarding the computed solution can be gained by looking at the norm of the gradient (which we know to ultimately go to zero). We have (again according to Prop. 2):

$$\|\nabla R_N(\psi)\|_N^2 = \frac{1}{N} \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} \left(g_\psi(y|x_i) \Delta_\psi c_i(y) \right)^2.$$

For the gradient to be small, each term $\sum_{y\in\mathcal{Y}}(g_\psi(y|x_i)\Delta_\psi c_i(y))^2$ has to be small. Let fix x_i. This in turn means that for any y, either $\Delta_\psi c_i(y)$ is small, or $g_\psi(y|x_i)$ is small. For simplicity, we identify here $\mathcal{Y}$ with $\{1,\ldots,K\}$. For the cost-insensitive multi-class classification case, and without loss of generality, assume that the correct label is 1:

$$c_i = \begin{pmatrix} 0 & 1 & \ldots & 1 \end{pmatrix}^\top.$$

Let $\eta_1 = g_\psi(1|x_i)$ be the probability of choosing the right label (and similarly, $\eta_j = g_\psi(j|x_i)$). In this case, the expected cost is $\sum_{y\in\mathcal{Y}} g_\psi(y|x_i)c_i(y) = 1-\eta_1$. As a result, we have $\Delta_\psi c_i(1) = -(1-\eta_1)$ and $\Delta_\psi c_i(k>1) = \eta_1$. Consequently, we can write

$$\sum_{y\in\mathcal{Y}}(g_\psi(y|x_i)\Delta_\psi c_i(y))^2 = \eta_1^2\left((1-\eta_1)^2 + \sum_{j=2}^{K}\eta_j^2\right).$$

For this quantity to be small, we should have either $\eta_1 \approx 1$ or $\eta_1 \approx 0$. In the first case, the right label is chosen. The second case is more problematic: the probabilities are distributed among any combination of labels, except the good one. However, let ψ' be a function in the neighborhood of ψ, such that $\eta_1' = \eta_1 + \epsilon$, all other things being equal (notably, the probabilities do not change for other inputs). Then, we have that

$$R_N(\psi') = R_N(\psi) - \frac{\epsilon}{N}.$$

This means that ψ cannot be a local minimum, as an arbitrarily small perturbation can lead to a lower risk. This let hope that $\eta_1 \approx 1$ is more likely to happen and that sm-boost computes a good solution. This has to be confirmed experimentally.

5 Experimental results

In this section, we provide experimental results on both synthetic (Sec. 5.3) and real-world (Sec. 5.4) data sets. The proposed approach will be compared to AdaBoost.MH [22], a state-of-the-art boosting algorithm that uses the same type of base learner, and to SAMME [26] (Stagewise Additive Modeling using a Multi-Class Exponential loss function), a more recent multi-class extension of AdaBoost, to be briefly presented in Sec. 5.2. It will appear that sm-boost is less sensitive to noise. Yet, it is known that convex boosters (what are AdaBoost.MH and SAMME) are not noise-tolerant [16]. Therefore, sm-boost will be compared to another non-convex booster, MartiBoost [15] (Martingale Boosting) on a challenging toy problem [15] in Sec. 5.5. First, we discuss the choice of meta-parameters for sm-boost.

5.1 Meta-parameters

There are three meta-parameters to be chosen in sm-boost: the base learner ($\mathcal{H}$), the learning rate (α_i) and the number of samples to be drawn for each projection (M).

When using binary classifiers as base learners, sm-boost reduces (cost-sensitive) multi-class classification to weighted binary classification. However, the inputs of those intermediate steps are a concatenation of original inputs and labels, sampled in the set $\{(x_i, y)_{1\leq i\leq N, y\in\mathcal{Y}}\}$. There are multiple solutions to this. For example, labels can be considered as they are, or they can be binarized. However, a standard classifier will more likely handle inputs from $\mathcal{X}$, the original input set. Let $\mathcal{F}\subset\{-1,+1\}^{\mathcal{X}}$ be the hypothesis space generated by such a base learner, we will consider $\mathcal{H}=\mathcal{F}^{\mathcal{Y}}$. More precisely, writing δ_y the Dirac in $y\in\mathcal{Y}$, we consider:

$$\mathcal{H}=\{h(x,y)=\sum_{z\in\mathcal{Y}}\delta_z(y)f_z(x) \text{ such that } \forall z\in\mathcal{Y},\ f_z\in\mathcal{F}\}.$$

The binary classification objective function (8) can thus be written as:

$$R_M(h)=\frac{1}{M}\sum_{j=1}^{M}\Delta_\psi c_j(z_j)h(x_j,z_j)=\sum_{y\in\mathcal{Y}}R_{M,y}(f_y)$$
$$\text{with } R_{M,y}(f)=\frac{1}{M}\sum_{j=1}^{M}\delta_y(z_j)\Delta_\psi c_j(z_j)f(x_j).$$

Therefore, computing h^* amounts to solving K weighted binary classification problems:

$$\operatorname*{argmax}_{h\in\mathcal{H}}R_M(h)=\left(\operatorname*{argmax}_{f\in\mathcal{F}}R_{M,y}(f)\right)_{y\in\mathcal{Y}}.$$

There remains to choose $\mathcal{F}$. In all forthcoming experiments, we consider classification trees (the CART algorithm [3] provided in the `scikit-learn` library [19]), growth in a best-first fashion to a maximum of 12 leafs.

The second meta-parameter to be chosen is the learning rate. Recall that Th. 1 holds for the simple choice $\alpha_t=1$, for all $t>0$. Yet, we will practically consider a more aggressive (yet still simple) learning rate. Recall that the "global" learning rate is

$$\alpha_t\frac{\langle\nabla R_N(\psi_{t-1}),h_t^*\rangle_N}{\|h_t^*\|_N^2}.$$

First, for binary classifiers, we have $\|h_t^*\|_N^2=K$. In order to be insensitive to the number of classes, we choose α_t to be proportional to the number of classes. Second, recall that theoretically h_t^* is computed to maximize $\langle\nabla R_N(\psi_{t-1}),h\rangle_N$, which is indeed an expectation, see Eq. (7). In order to introduce randomization, we have replaced this expectation by the Monte Carlo

estimate R_M (see Sec. 3.3). Therefore, it is natural to consider $R_M(h_t^*)$ instead of $\langle \nabla R_N(\psi_{t-1}), h_t^* \rangle_N$ (this initial inner product being somehow more conservative). To sum up, in all forthcoming experiments the learning rate will be

$$\alpha_t = K \frac{R_M(h_t^*)}{\langle \nabla R_N(\psi_{t-1}), h_t^* \rangle_N}$$
$$\Leftrightarrow \alpha_t \frac{\langle \nabla R_N(\psi_{t-1}), h_t^* \rangle_N}{\|h_t^*\|_N^2} = R_M(h_t^*) = \frac{1}{M} \sum_{j=1}^{M} \Delta_{\psi_{t-1}} c_j(z_j) h_t^*(x_j, z_j),$$

with $(x_j, z_j)_{1 \leq j \leq M}$ the samples drawn at the t^{th} iteration.

The last meta-parameter is M, the number of samples to be drawn for training the weak learner at each iteration. With a default initialization $\psi_0 = 0$, the initial decision rule is uniformly random and NK samples are candidate for sampling. Asymptotically, the decision rule is likely to become deterministic (at least for single-label multi-class classification, as seen in Sec. 4), so there will be N candidate samples for sampling. Therefore, $M = N$ seems to be a good default choice, corresponding roughly to an adaptive subsampling. In all forthcoming experiments, we set $M = N$.

5.2 Competitors

We present briefly the algorithms to which sm-boost will be compared, namely AdaBoost.MH [22] and SAMME [26]. The presentation of MartiBoost [15], that can handle only binary classification problems, is postponed to Sec. 5.5.

5.2.1 The AdaBoost.MH algorithm

AdaBoost, introduced in [6], is the first boosting algorithm, dedicated to the binary classification problem. It has been shown [17,7] that AdaBoost is indeed a functional gradient descent minimizing the expected exponential loss (so, a convex surrogate to the risk based on the binary loss), with a learning rate computed using an exact line search (which is analytically possible in this case, contrary to sm-boost). AdaBoost.MH [22] is a multi-class generalization of AdaBoost based on Hamming loss.

There are indeed many generalizations of AdaBoost to the multi-class setting. However, AdaBoost.MH provides competitive results [22,13] and it has been argued that with large samples, it has optimal classification performance [25]. Moreover, the underlying mechanistic of sm-boost is very close to the one of AdaBoost.MH (training weak learners on re-weighted samples of the original data set) and both algorithms are based on the same type of weak learners ($\mathcal{H} \subset \{-1, +1\}^{\mathcal{X} \times \mathcal{Y}}$), which makes comparisons easier and more reasonable.

AdaBoost.MH is summarized in Alg. 3, using the same notations as sm-boost to ease the comparison. Here, the cost c belongs to $\{-1,+1\}^{\mathcal{X}\times\mathcal{Y}^2}$ (instead of $\{0,+1\}^{\mathcal{X}\times\mathcal{Y}^2}$ for the binary loss considered so far):

$$c(x_i,y_i,y) = 2\mathbb{I}_{y\neq y_i} - 1 \text{ (instead of } \mathbb{I}_{y\neq y_i}).$$

The algorithm also maintains a distribution w used to train successive weak learners, this distribution being updated according to the related results.

Algorithm 3: AdaBoost.MH

Inputs;
Data set $\{(x_i,y_i)_{1\leq i\leq N}\}$;
Weak learner ($\mathcal{H}$) ;

Initialization;
Initialize $\psi_0 = 0$;
Set the initial distribution: $w_0(x_i,y) = \frac{1}{NK}$, $1\leq i\leq N$, $y\in\mathcal{Y}$;

for $t = 1,2,\dots T$ **do**

Solve h_t^* as the solution of the weighted binary classification problem with ;
inputs: $(x_i,y)_{1\leq i\leq N, y\in\mathcal{Y}}$;
weights: $(w_{t-1}(x_i,y))_{1\leq i\leq N,y\in\mathcal{Y}}$;
outputs: $(-c_i(y))_{1\leq i\leq N,y\in\mathcal{Y}}$;

Compute the edge γ_t and the learning rate α_t ;
$\gamma_t = -\sum_{i=1}^N \sum_{y\in\mathcal{Y}} w_{t-1}(x_i,y) h_t^*(x_i,y) c_i(y)$;
$\alpha_t = \frac{1}{2}\ln\frac{1+\gamma_t}{1-\gamma_t}$;

Update the distribution w, for all $1\leq i\leq N$, $y\in\mathcal{Y}$;
$w_t(x_i,y) = \frac{w_{t-1}(x_i,y) e^{\alpha_t c_i(y) h_t^*(x_i,y)}}{\sum_{j=1}^N \sum_{z\in\mathcal{Y}} w_{t-1}(x_j,z) e^{\alpha_t c_j(z) h_t^*(x_j,z)}}$;

Update the score function ;
$\psi_t = \psi_{t-1} + \alpha_t h_t^*$;

Output the final hypothesis ;
the deterministic decision rule is $x \to \text{argmax}_{y\in\mathcal{Y}}(\text{sgn}(\psi_T(x,y)))$;

There are strong similarities between AdaBoost.MH and sm-boost. Both algorithms re-weight samples of the original data set to train similar weak classifiers. Consider the (global) learning rate of sm-boost,

$$\frac{1}{M}\sum_{j=1}^M \Delta_{\psi_{t-1}} c_j(z_j) h_t^*(x_j,z_j),$$

it is also an edge (as the sum of weighted prediction-target products). Moreover, for a small edge γ (one can easily show that $\gamma\in]0,1]$ for a weak binary classifier doing better than random guessing), one has $\frac{1}{2}\ln\frac{1+\gamma}{1-\gamma} = \gamma + o(\gamma^2)$. Therefore, both algorithms share a similar learning rate when the edge is small. There are also strong differences. The weighting scheme, as well as the targets

for intermediate binary classifiers, are different (the target being adaptive for sm-boost, contrary to AdaBoost.MH), and they output different decision rule (respectively deterministic and stochastic). This was to be expected, as both approaches attempt to minimize different risks.

The only meta-parameter of AdaBoost.MH is the base learner. In all forthcoming experiments, we use the same weak learner as for sm-boost. Notice that AdaBoost.MH has a higher computational cost: sm-boost requires training K classifiers with $M = N$ samples divided up among those classifiers, while AdaBoost.MH requires training K classifiers with N samples each.

5.2.2 The SAMME algorithm

SAMME [26] is a more recent multi-class extension of AdaBost, that does no reduce to multiple two-class problems. It is obtained as a forward stagewise additive modeling approach that minimizes an exponential loss for multi-class classification. In other words, the considered weak learners have to solve weighted multi-class classification problems. SAMME is summarized in Alg. 4 (and is exactly AdaBoost for the case $K = 2$).

Algorithm 4: SAMME

Inputs;
Data set $\{(x_i, y_i)_{1 \leq i \leq N}\}$;
Weak learner ($\mathcal{H}$) ;

Initialization;
Initialize $\psi_0 = 0$;
Set the initial distribution: $w_0(x_i) = \frac{1}{N}$, $1 \leq i \leq N$;

for $t = 1, 2, \dots T$ **do**

Solve h_t^* as the solution of the weighted *multi-class* classification problem with ;
inputs: $(x_i)_{1 \leq i \leq N}$;
weights: $(w_{t-1}(x_i))_{1 \leq i \leq N}$;
outputs: $(y_i)_{1 \leq i \leq N}$;

Compute the error ϵ_t and the learning rate α_t ;
$\epsilon_t = \sum_{i=1}^N w_{t-1}(x_i) \mathbb{I}_{\{y_i \neq h_t^*(x_i)\}}$;
$\alpha_t = \ln \frac{1-\epsilon_t}{\epsilon_t} + \ln(K-1)$;

Update the distribution w, for all $1 \leq i \leq N$;
$$w_t(x_i) = \frac{w_{t-1}(x_i) e^{\alpha_t \mathbb{I}_{\{y_i \neq h_t^*(x_i)\}}}}{\sum_{j=1}^N w_{t-1}(x_j) e^{\alpha_t \mathbb{I}_{\{y_j \neq h_t^*(x_j)\}}}} ;$$

Update the score function ;
$\psi_t(x, y) = \psi_{t-1}(x, y) + \alpha_t \mathbb{I}_{\{h_t^*(x) = y\}}$;

Output the final hypothesis ;
the deterministic decision rule is $x \to \text{argmax}_{y \in \mathcal{Y}}(\psi_T(x, y))$;

The only meta-parameter of SAMME is the base learner. As for sm-boost and AdaBoost.MH, we consider classification trees. Yet, recall that these al-

gorithms have to train one weak learner per class (with tress growth in a best-first fashion to a maximum of 12 leafs), while SAMME trains only one weak learner per iteration (that performs multi-class classification rather than binary classification). Therefore, to have a fair comparison, the base learner is here a classification tree growth to a maximum of $(K-1)\cdot 12$ leafs[1] .

5.3 Synthetic problems

The first considered experiment are synthetic and randomly generated multi-class classification problems, inspired from [10] (Madelon data set) and whose implementation is provided in [19]. Sampling such a problem is parameterized by d_{info} (the number of informative features), d_{red} (the number of redundant features that are random linear combination of informative ones), d_{rep} (the number of repeated features), d_{useless} (the number of useless features, drawn at random), K (the number of classes), C_K (the number of cluster per classes), β (the fraction of samples whose classes are randomly exchanged) and N (the number of training samples). With these parameters, a classification problem is sampled by creating clusters of points normally distributed (standard deviation of one) about vertices of a d_{info}-dimensional hypercube and assigning an equal number of cluster to each class. Interdependences between these features are introduced and various type of further noise are added to the data, as described above. Moreover, the features are shifted by a random value drawn in $[-1,1]$ and scaled by another random value drawn in $[1,100]$. This provide a classification problem with $d = d_{\text{info}} + d_{\text{red}} + d_{\text{rep}} + d_{\text{useless}}$ features and K classes.

5.3.1 Comparison to AdaBoost.MH

We generated 1000 such problems with $d_{\text{info}} \sim \mathcal{U}_{[2,10]}$ ($\mathcal{U}$ denotes the uniform distribution), $d_{\text{red}} \sim \mathcal{U}_{[0,3]}$, $d_{\text{rep}} \sim \mathcal{U}_{[0,3]}$, $d_{\text{useless}} \sim \mathcal{U}_{[0,3]}$, $K \sim \mathcal{U}_{[2,5]}$, $C_K \sim \mathcal{U}_{[2,5]}$, $\beta \sim \mathcal{U}_{[0,0.25]}$ and $N \sim \mathcal{U}_{[300,3000]}$. For each problem, we also sampled $N_{\text{test}} = 10000$ data points (with noiseless labels) to estimate the true risk (so, not used in the training set, of which we recall the size to be drawn randomly). For each such problem, we trained AdaBoost.MH and sm-boost for $T = 1000$ iterations.

By training error (an error being denoted generically ϵ in the following), we mean the empirical risk (with the binary loss) on training data, and by testing (or generalization) error we mean the estimation of the true risk (still

[1] For the stochastic decision rule used in sm-boost, we have that, for any $\varphi \in \mathbb{R}^{\mathcal{X}}$,

$$g_\psi(y|x) = \frac{e^{\psi(x,y)}}{\sum_{z\in\mathcal{Y}} e^{\psi(x,z)}} = \frac{e^{\psi(x,y)+\varphi(x)}}{\sum_{z\in\mathcal{Y}} e^{\psi(x,z)+\varphi(x)}}.$$

Therefore, there is some redundancy among the K trained weak classifiers, and there is indeed $K-1$ degrees of liberty. That explains the choice of $(K-1)\cdot 12$ leafs (rather than $K\cdot 12$ leafs).

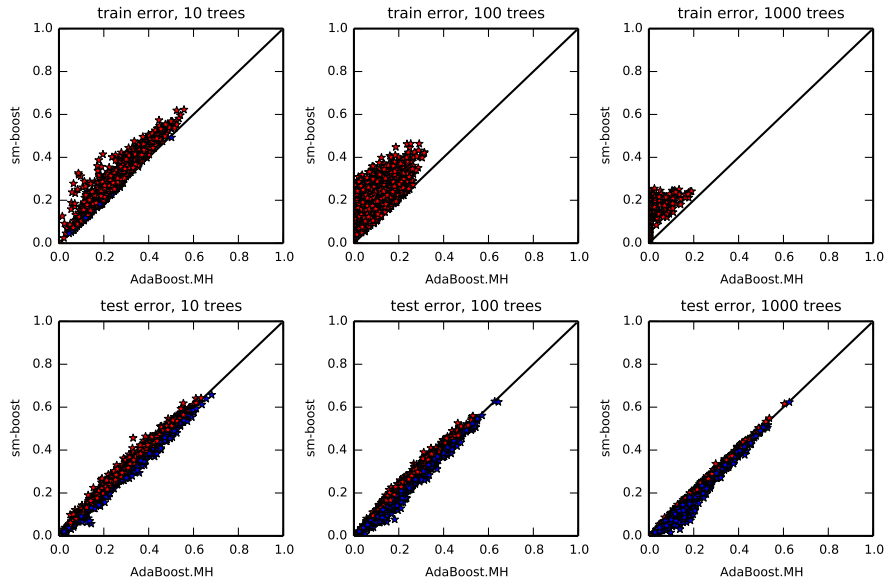

Fig. 1 Synthetic problems, AdaBoost.MH versus sm-boost.

with the binary loss, with noiseless labels here) based on testing data (sampled independently). In Fig. 1, we compare for both algorithms the training error (first row) and the testing error (second row) after respectively 10, 100 and 1000 iterations (first to third column). Each graph provides $\epsilon_{\text{AdaBoost.MH}}$ in abscissa and $\epsilon_{\text{sm-boost}}$ in ordinate. A point above the line $y = x$ (in red) means that $\epsilon_{\text{AdaBoost.MH}} < \epsilon_{\text{sm-boost}}$ (AdaBoost.MH is better), and a point under the line (in blue) means that $\epsilon_{\text{AdaBoost.MH}} > \epsilon_{\text{sm-boost}}$ (sm-boost is better).

One can see on Fig. 1 that AdaBoost.MH provides almost always a lower training error. This might appear surprising, as sm-boost directly attempt to minimize this error, contrary to AdaBoost.MH. However, the exponential loss is a good surrogate to the binary one, the underlying optimization problem is convex (not the case for sm-boost), and the learning rate is computed by an exact line search (again, not the case for sm-boost). So, this is not such a big surprise. Yet, regarding the generalization error, sm-boost is competitive, and often better than AdaBoost.MH for large number of iterations (despite the lack of convexity and of an optimal learning rate).

To get more insights into the difference of performance, Fig. 2 provides histograms of the difference of errors, $\epsilon_{\text{AdaBoost.MH}} - \epsilon_{\text{sm-boost}}$, for the same cases (training and testing error for $T = 10$, 100 and 1000 iterations). This confirms the above observations.

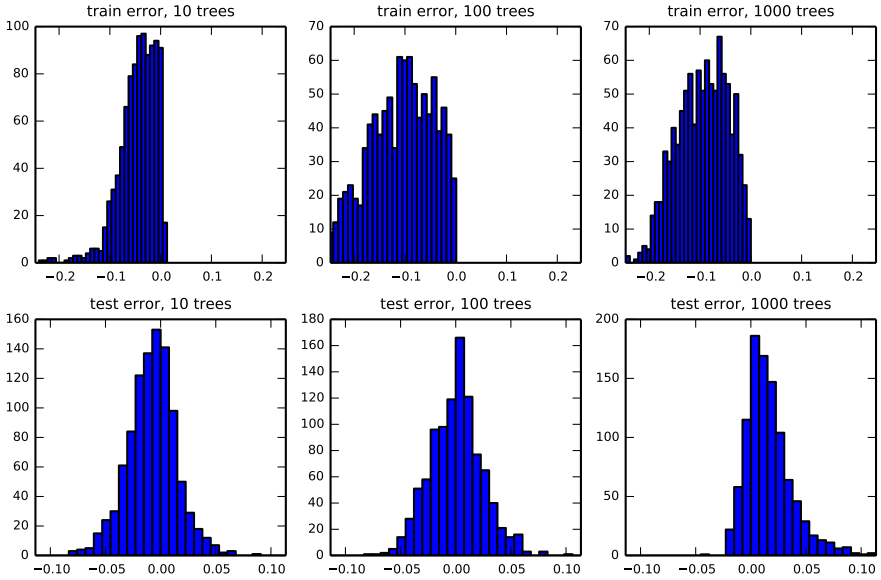

Fig. 2 Synthetic problems, histograms of $\epsilon_{\text{AdaBoost.MH}} - \epsilon_{\text{sm-boost}}$.

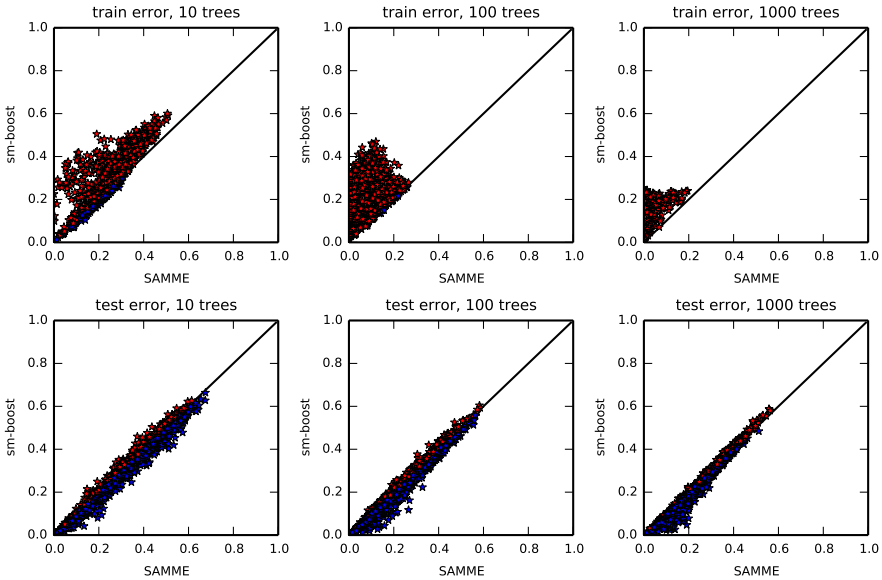

Fig. 3 Synthetic problems, SAMME versus sm-boost.

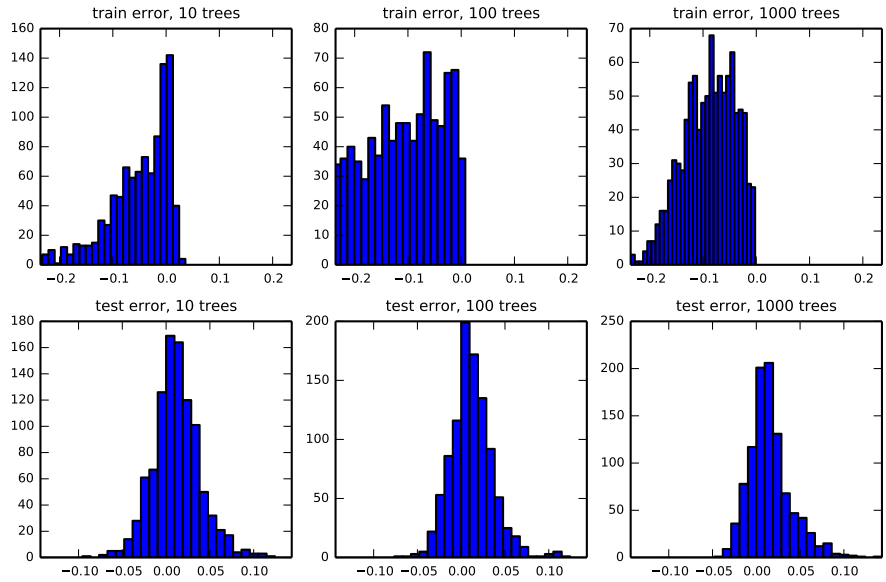

Fig. 4 Synthetic problems, histograms of $\epsilon_{\text{SAMME}} - \epsilon_{\text{sm-boost}}$.

5.3.2 Comparison to SAMME

The same experience has been performed to compare SAMME and sm-boost (1000 problems were generated, with the same random laws for drawing problems meta-parameters, and both algorithms were trained for $T = 1000$ iterations, for each random problem). Results are provided in Fig. 3 (comparison of training and testing errors after respectively 10, 100 and 1000 iterations) and Fig. 4 (histograms of the difference of errors). The same conclusions can be drawn (higher training error and lower generalization error for sm-boost).

5.4 Real-world data sets

In this section, we show the results of running sm-boost on a collection of data sets from the libsvm repository [2] (these data sets coming from various repositories, such as UCI or Statlog for example). Each data set comes with pre-specified training and testing sets and is summarized in Table 1.

We compare AdaBoost.MH, SAMME and sm-boost using these pre-specified sets, and provide results after respectively $T = 10$, $T = 100$ an $T = 1000$ iterations (or trees) in Table 2. It can be seen that AdaBoost.MH or SAMME almost always provide a lower error, and that the difference of errors tends to decrease with the number of iterations (but still in disfavor of sm-boost).

[2] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

Table 1 Benchmark data sets.

Data set	# train	# test	# features	# classes
dna	2000	1186	180	3
letter	15000	5000	16	26
mnist	60000	10000	784	10
pendigits	7494	3498	16	10
satimage	4435	2000	36	6
segment	210	2100	19	10
splice	1000	2175	60	2
vowel	528	462	10	11

Table 2 Test error (%) rates on considered data sets.

		$T = 10$	$T = 100$	$T = 1000$
dna	AdaBoost.MH	**5.14**	**3.96**	**4.05**
	SAMME	6.15	4.64	4.46
	sm-boost	6.49	4.89	4.13
letter	AdaBoost.MH	15.0	3.72	**2.34**
	SAMME	**13.0**	**3.68**	2.84
	sm-boost	26.9	10.7	5.40
mnist	AdaBoost.MH	12.7	4.14	**1.76**
	SAMME	**10.8**	**3.72**	2.52
	sm-boost	14.0	7.12	3.56
pendigits	AdaBoost.MH	4.86	**2.43**	**2.17**
	SAMME	**3.03**	2.49	2.29
	sm-boost	8.23	4.57	2.83
satimage	AdaBoost.MH	13.0	10.0	9.20
	SAMME	13.7	**9.85**	**8.8**
	sm-boost	**12.9**	10.8	8.95
segment	AdaBoost.MH	**5.90**	**6.00**	6.38
	SAMME	10.86	10.86	10.86
	sm-boost	9.19	6.95	**6.19**
splice	AdaBoost.MH	**2.94**	**2.71**	**2.53**
	SAMME	5.75	3.17	2.71
	sm-boost	4.64	3.63	3.08
vowel	AdaBoost.MH	49.4	44.6	42.9
	SAMME	**47.8**	**39.4**	**38.3**
	sm-boost	56.9	45.7	46.3

To get a larger picture of what happens for this testing error (as well as for the learning one), we ran 25 independent runs (shuffled data, using the same train and test sizes). Fig. 5 provides the learning (left subfigures) and testing (right) errors (mean $\pm$ standard deviation represented) as a function of the number of iterations, for each data set. The trend of Table 2 is confirmed: consistently lower AdaBoost.MH or SAMME error and usually diminishing difference of errors. This also provides some insight on the learning error: the ones of AdaBoost.MH and SAMME diminishe more rapidly to a lower value.

Recall the results of the synthetic experiments (Sec. 5.3). If learning errors have a similar behavior, it is not the case for testing errors (sm-boost is generally better, for a large number of iterations, in the synthetic experiments).

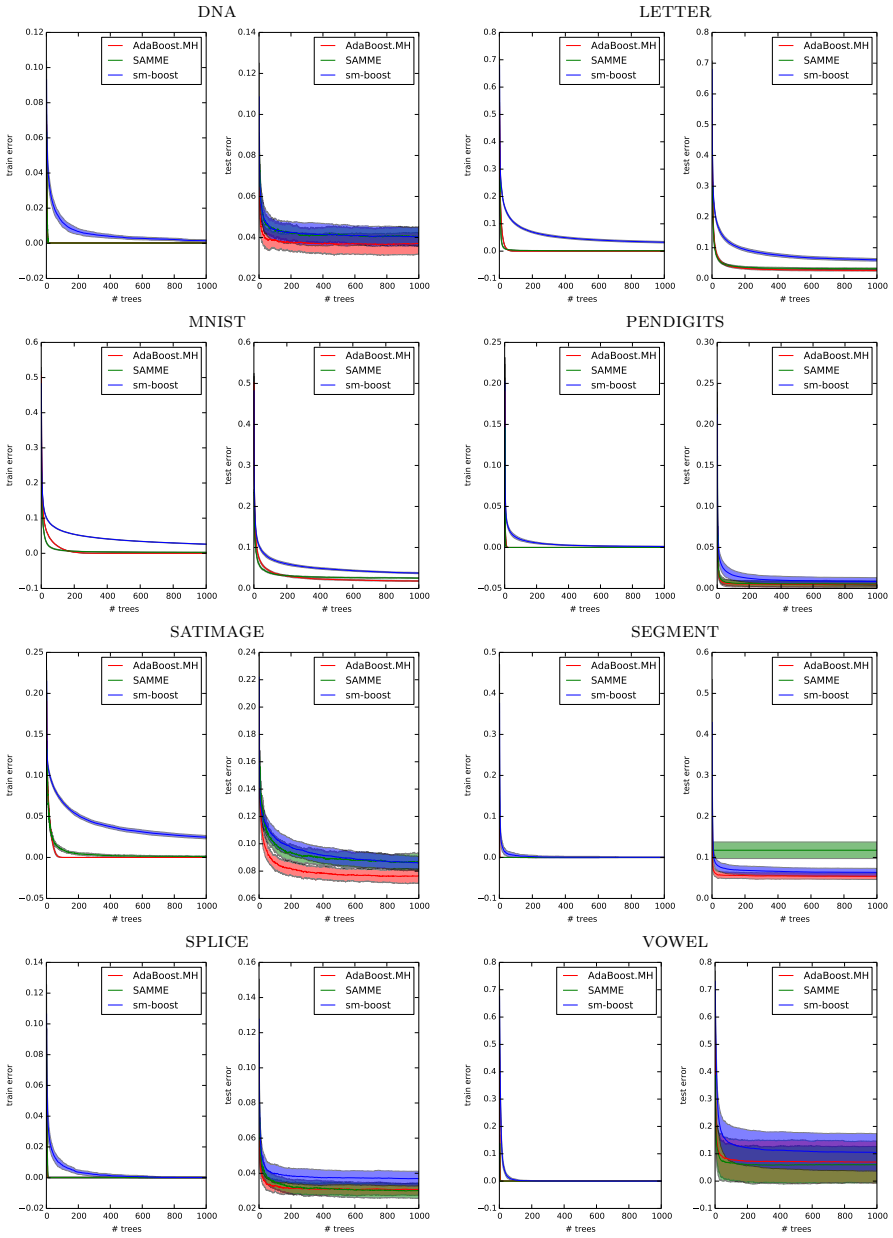

Fig. 5 Train and test error curves for the considered data sets.

Table 3 Test error (%) rates on considered data sets with corrupted labels (20% of samples have randomly exchanged labels). In parentheses we recall the noiseless results of Table 2.

		$T = 10$	$T = 100$	$T = 1000$
dna	AdaBoost.MH	8.26 (5.14)	7.67 (3.96)	7.25 (4.05)
	SAMME	13.8 (6.15)	10.5 (4.64)	8.85 (4.46)
	sm-boost	**6.15** (6.49)	**4.64** (4.89)	**6.24** (4.13)
letter	AdaBoost.MH	26.2 (15.0)	15.1 (3.72)	8.58 (2.34)
	SAMME	**25.8** (13.0)	**11.6** (3.68)	**6.84** (2.84)
	sm-boost	29.2 (26.9)	13.2 (10.7)	6.98 (5.40)
mnist	AdaBoost.MH	19.8 (12.7)	11.5 (4.14)	7.01 (1.76)
	SAMME	17.4 (10.8)	12.1 (3.72)	5.48 (2.52)
	sm-boost	**14.1** (14.0)	**6.96** (7.12)	**3.78** (3.56)
pendigits	AdaBoost.MH	10.9 (4.86)	7.63 (2.43)	5.77 (2.17)
	SAMME	11.7 (3.03)	6.66 (2.49)	4.49 (2.29)
	sm-boost	**9.06** (8.23)	**3.97** (4.57)	**3.06** (2.83)
satimage	AdaBoost.MH	16.7 (13.0)	12.8 (10.0)	10.9 (9.20)
	SAMME	17.3 (13.7)	12.9 (9.85)	10.7 (8.8)
	sm-boost	**13.8** (12.9)	**11.8** (10.8)	**10.4** (8.95)
segment	AdaBoost.MH	**15.6** (5.90)	13.8 (6.00)	13.9 (6.38)
	SAMME	17.0 (10.86)	14.0 (10.86)	13.0 (10.86)
	sm-boost	15.9 (9.19)	**12.4** (6.95)	**12.0** (6.19)
splice	AdaBoost.MH	16.9 (2.94)	16.7 (2.71)	14.9 (2.53)
	SAMME	18.3 (5.75)	17.2 (3.17)	14.3 (2.71)
	sm-boost	**9.15** (4.64)	**10.7** (3.63)	**13.1** (3.08)
vowel	AdaBoost.MH	55.8 (49.4)	47.2 (44.6)	47.6 (42.9)
	SAMME	54.1 (47.8)	**44.6** (39.4)	**41.3** (38.3)
	sm-boost	**52.8** (56.9)	48.9 (45.7)	43.7 (46.3)

However, recall that various type of noises were introduced in the data in Sec. 5.3, which is not the case for real data (at least, not willingly).

Therefore, we corrupted the real-world data sets by randomly exchanging labels of 20% of samples. We provide results after respectively $T = 10$, $T = 100$ an $T = 1000$ iterations (or trees) in Table 3, recalling in parentheses the noiseless results from Table 2. We also provide a larger picture of what happens with Fig. 6 (similar to Fig. 5, but with noisy data). The trend regarding the learning errors is the same, but the trend regarding the testing error is reversed: now, sm-boost provides almost always lower errors.

To summarize these experimental results, sm-boost appears to be competitive with AdaBoost.MH and SAMME. It provides higher generalization errors for the considered standard real-world data set, but it is more robust to noisy data (real data sets and synthetic experiments). Also, there might exist better choices for the learning rate or the number of samples to be drawn at each iteration, for sm-boost. This algorithm is also slower to train (as seen on the train error curves), and increasing the number of iterations could improve the generalization error. If sm-boost is clearly not the ultimate classification algorithm, these experiments show that minimizing directly the risk of interest is a viable alternative to more standard approaches.

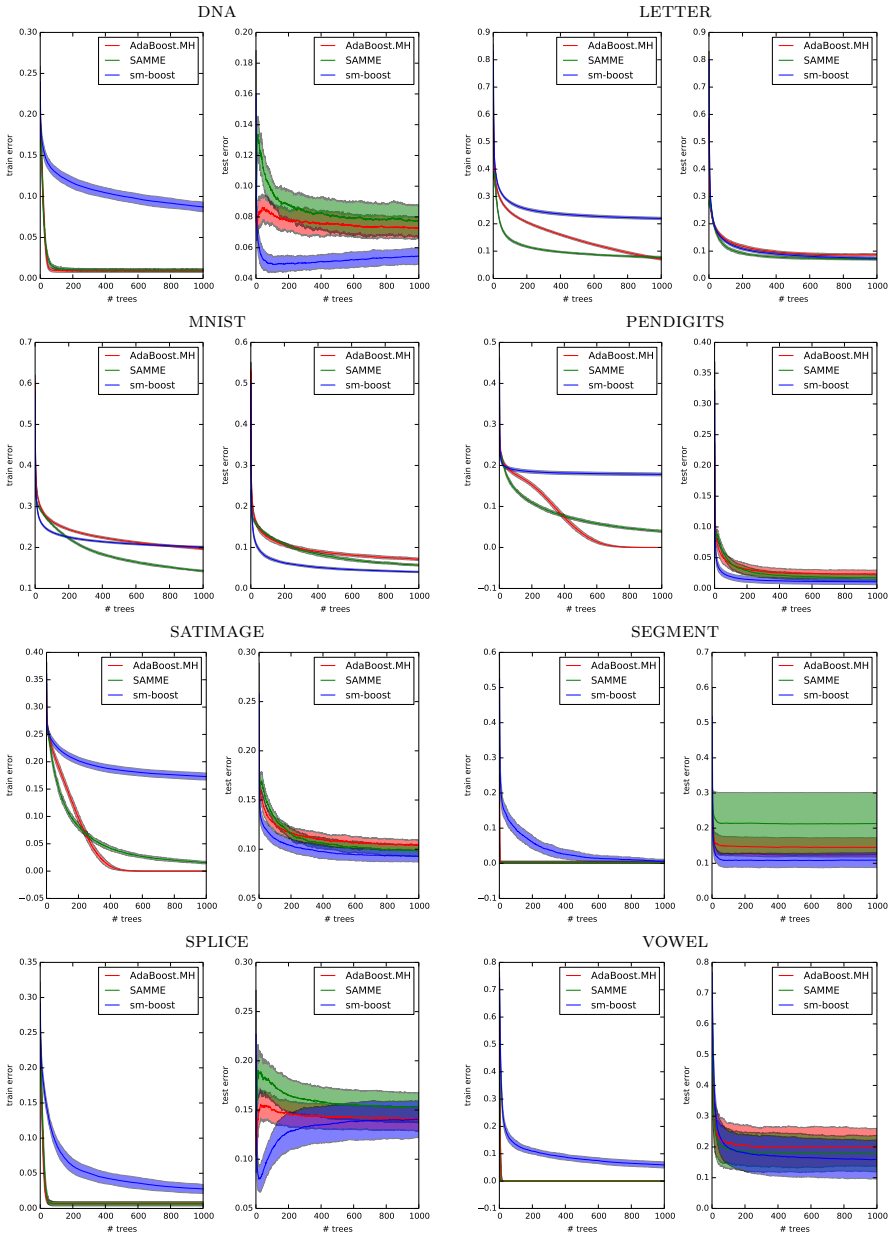

Fig. 6 Train and test error curves for the considered data sets, with corrupted labels.

5.5 About non-convex boosters

Previous experiments have shown that sm-boost is less sensitive to noise than AdaBoost.MH and SAMME. However, it has been shown in [16] that convex potential boosters (what are AdaBoost and its generalizations, among others, and what is not sm-boost) are defeated by random classification noise. There exist non-convex boosters (*e.g.* [4,5,11,15]). Yet, as far as we know, these algorithms are dedicated to the cost-insensitive binary classification problem, and cannot be extended easily to the cost-sensitive multi-class problem. Nevertheless, we compare in this section sm-boost to another non-convex booster, MartiBoost [15] (Martingale Boosting), that comes with theoretical guarantees (notably regarding tolerance to noise). Algorithms (sm-boost, MartiBoost and SAMME, that results in AdaBoost in the binary case) will be compared on the challenging (for convex problem) toy problem proposed in [16].

5.5.1 Martingale Boosting

MartiBoost [15] constructs a branching program with $L+1$ layers in a grid graph structure, where layer l has l nodes (see Fig. 7, left). Each node of the first L layers corresponds to a weak learner (therefore, there is a total of $T = \frac{L(L+1)}{2}$ weak learners). Once trained, prediction is done as follows. A new input x is presented to the top root node. The corresponding weak learner predicts the label, $+1$ or 0, which results in routing the example x respectively to the right or to the left in the next layer. This procedure is repeated until the sample reaches the last layer. If the last node is in the right part of the last layer (if its index is greater than $\frac{L}{2}$), then the final prediction is $+1$, else it is 0.

Training such a branching program assumes the availability of a two-sided weak learner (roughly, a weak learner that provides the same accuracy for both positive and negative examples). The program is trained layer by layer. The empirical distribution (corresponding to the training set) is filtered until reaching an untrained node (samples being routed through already trained nodes). Then this filtered distribution is used to train the two-sided weak learner. However, usual weak learners are one-sided (they try to minimize the expected binary loss). Yet, such a weak learner can be used to simulate a two-sided one. To do so, the filtered distribution is balanced (such that positive and negative samples have the same cumulated probability masses) before training, and the resulting decision rule is randomized (such that it outputs each label equally often). For more details, see [15].

5.5.2 Considered toy problem

The toy problem we consider has been proposed in [16]. Each training set consists of 4000 samples, divided into three groups: 1000 "large margin examples", 1000 "pullers" and 2000 "penalizers". Each labeled example (x, y) in the set is generated as follows. The label y is chosen randomly in $\{0, 1\}$ (with equal

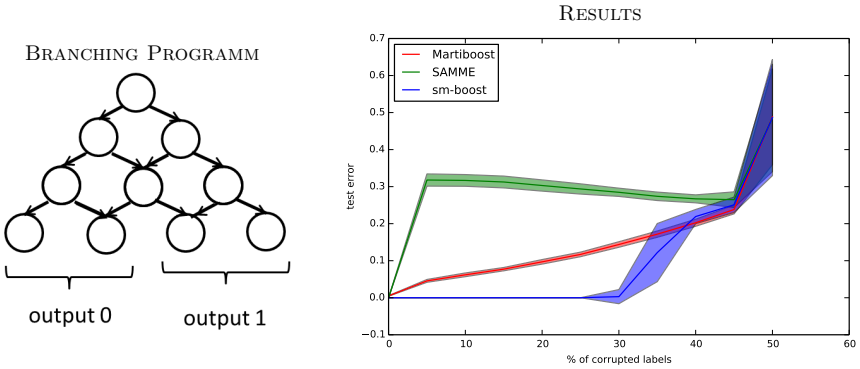

Fig. 7 (Left) the branching programm constructed by MartiBoost. (Right) results on the challenging noisy toy problem.

probability). The input x is a 21-dimensional binary vector with components $x_1, \ldots, x_{21}$. Each large margin example assigns $x_1 = \cdots = x_{21} = y$. Each puller sets $x_1 = \cdots = x_{11} = y$ and $x_{12} = \cdots = x_{21} = 1 - y$. Penalizers are chosen at random in three stages: the values of a random subset of five of the first eleven components are set to y, the values of a random subset of six of the last ten components are also set to y, and the remaining ten components are set to $1-y$. If base learners are decision stumps associated to each component, a majority vote over them leads to a correct classification of all samples. Yet, noise is added in the training set: each label y is corrupted with classification noise (that is, set to $1 - y$) with probability p (with $0 \leq p \leq 0.5$). Eventually, each testing set consists of 10000 samples (2500 large margin examples, 2500 pullers and 5000 penalizers), with uncorrupted labels.

5.5.3 Results

We compare sm-boost to SAMME (so, AdaBoost in this binary case) and MartiBoost, for varying levels of noise (p varying from 0 to 0.5 by steps of 0.05). For all boosters, weak learners are decision stumps (classification trees with two leafs). SAMME and sm-boost are trained for $T = 1000$ iterations (thus $T = 1000$ weak learners). MartiBoost is trained for $L = 300$ stages (thus $T = 45150$ weak learners). For each level of noise, the experiment (generating a training set, as depicted before, to train all algorithms, and generating a testing set to evaluate them) is repeated 100 times. Results are presented in Fig. 7 (right), that shows for each algorithm the test error after training as function of the noise level (mean $\pm$ standard deviation represented).

Without noise, all boosters have a perfect accuracy. SAMME's performance degrades with as low as 5% of noise (and this result is consistent with the one in [16]). MartiBoost accuracy decreases when the level of noise increases. Yet, the test error always remains below the noise level. On the other hand, sm-boost has a perfect accuracy for a large range of noise, that deteriorates to reach the accuracy of MartiBoost after 40% of corrupted labels.

6 Conclusion

This paper has introduced sm-boost, a boosting algorithm which directly minimizes the expected binary loss, considering a stochastic decision rule parameterized by a soft-max conditional distribution. Its convergence has been analyzed and discussed. The proposed approach has also been compared empirically to AdaBoost.MH and SAMME, and has been shown to be competitive, especially in the case of noisy data (AdaBoost.MH and SAMME remaining more efficient, regarding both learning and testing errors, for clean data). As it is known that convex boosters are sensitive to noise, we also compared sm-boost to another non-convex booster, MartiBoost. The proposed algorithm appears to be more robust to noise.

As said before, we argue that minimizing directly the risk of interest is a viable alternative to more standard approaches. It is all the more true that some interesting perspectives exist. First, the proposed algorithm can be directly applied to cost-sensitive and multi-label multi-class classification (through the definition of the cost function c), which is not so straightforward for usual convex surrogates, as discussed in Sec. 2.2. Empirical evaluations on such problems is an interesting perspective. The analysis we provide is somehow weak, notably due to the lack of convexity; it would be interesting to study to what extent it could be improved (notably based on a margin point of view). Especially, two aspect should be studied: the quality of the reached local minimum (a question to which we provided an informal answer) and the generalization capability of the proposed estimator (empirical results show that sm-boost has a good generalization capability, this should be studied theoretically). The introduced risk is minimized using a naive (functional) gradient descent. Other (more efficient) optimization techniques may be envisioned. From a more practical point of view, the considered weak learners are quite simple (as we train K classifiers at each iteration). It has been shown that using such base learners does not provide the best results for AdaBoost.MH [13]. Therefore, sm-boost could be improved by using weak learners that take advantage of the particular inputs (original input-label tuple) [12,13]. Finally, extending this idea (minimizing directly the loss of interest through a stochastic decision rule) beyond boosting (for example to learning in Reproducing Kernel Hilbert Spaces) is worth being studied.

References

1. Bartlett, P.L., Jordan, M.I., McAuliffe, J.D.: Convexity, classification, and risk bounds. Journal of the American Statistical Association **101**(473), 138–156 (2006)
2. Beijbom, O., Saberian, M., Kriegman, D., Vasconcelos, N.: Guess-Averse Loss Functions For Cost-Sensitive Multiclass Boosting. In: International Conference on Machine Learning (ICML), pp. 586–594 (2014)
3. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and regression trees. CRC press (1984)
4. Freund, Y.: Boosting a weak learning algorithm by majority. Information and computation **121**(2), 256–285 (1995)

5. Freund, Y.: An adaptive version of the boost by majority algorithm. Machine learning **43**(3), 293–318 (2001)
6. Freund, Y., Schapire, R.E.: A desicion-theoretic generalization of on-line learning and an application to boosting. In: Computational learning theory, pp. 23–37. Springer (1995)
7. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. The annals of statistics **28**(2), 337–407 (2000)
8. Friedman, J.H.: Stochastic gradient boosting. Computational Statistics & Data Analysis **38**(4), 367–378 (2002)
9. Grubb, A., Bagnell, J.A.: Generalized boosting algorithms for convex optimization. In: International Conference on Machine Learning (ICML) (2011)
10. Guyon, I.: Design of experiments of the nips 2003 variable selection benchmark. In: NIPS 2003 workshop on feature extraction and feature selection (2003)
11. Kalaia, A.T., Servediob, R.A.: Boosting in the presence of noise. Journal of Computer and System Sciences **71**, 266–290 (2005)
12. Kégl, B., Busa-Fekete, R.: Boosting products of base classifiers. In: International Conference on Machine Learning (ICML), pp. 497–504. ACM (2009)
13. Kégl, B.: The return of AdaBoost.MH: multi-class Hamming trees. In: International Conference on Learning Representations (ICLR) (2014)
14. Lee, Y., Lin, Y., Wahba, G.: Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. Journal of the American Statistical Association **99**(465), 67–81 (2004)
15. Long, P.M., Servedio, R.A.: Martingale Boosting. In: Conference on Learning Theory (COLT), pp. 79–94. Springer-Verlag (2005)
16. Long, P.M., Servedio, R.A.: Random Classification Noise Defeats All Convex Potential Boosters. Machine Learning **78**(3), 287–304 (2010)
17. Mason, L., Baxter, J., Bartlett, P., Frean, M.: Boosting algorithms as gradient descent in function space. Tech. rep., Australian National University (1999)
18. Nesterov, Y.: Introductory lectures on convex optimization: A basic course. Springer (2004)
19. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
20. Pires, B.A., Ghavamzadeh, M., Szepesvári, C.: Cost-sensitive Multiclass Classification Risk Bounds. In: International Conference on Machine Learning (ICML) (2013)
21. Schapire, R.E., Freund, Y.: Boosting: Foundations and algorithms. MIT Press (2012)
22. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Machine learning **37**(3), 297–336 (1999)
23. Schölkopf, B., Smola, A.J.: Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press (2002)
24. Steinwart, I.: How to compare different loss functions and their risks. Constructive Approximation **26**(2), 225–287 (2007)
25. Zhang, T.: Statistical analysis of some multi-category large margin classification methods. The Journal of Machine Learning Research **5**, 1225–1251 (2004)
26. Zhu, J., Zou, H., Rosset, S., Hastie, T.: Multi-class AdaBoost. Statistics and Its Interface **2**, 249–360 (2009)